PARTICLE SWARM OPTIMIZATION

IN THE DYNAMIC ELECTRONIC WARFARE BATTLEFIELD

A Thesis

Submitted to the Faculty

of

Purdue University

by

Paul Ryan Witcher

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

May 2017

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Lauren Christopher, Chair

Department of Electrical and Computer Engineering

Dr. Brian King

Department of Electrical and Computer Engineering

Dr. Paul Salama

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King

Head of the Departmental Graduate Program

This thesis is dedicated ad maiorem Dei gloriam

ACKNOWLEDGMENTS

I would like thank Dr. Lauren Christopher, my major professor, for all of the guidance and support she has provided me with my research and thesis, along with the rest of my thesis committee, Dr. Brian King and Dr. Paul Salama. I would also like to thank the Crane Naval Surface Warfare Center for providing me an opportunity to participate in this exciting field of research. I also want to thank Dave Acton, Scot Hawkins, and James Stewart for all of the feedback and suggestions they have provided me.

Finally, I would like to thank my friends and family, especially my parents, Derrick and Kathryn; brother, Phil; roommate, Jaime Haro; and CRHP group for all of the help and support they have given me throughout this process.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

API	Application Programming Interface
EW	Electronic Warfare
GUI	Graphical User Interface
NSWC	Naval Surface Warfare Center
PSO	Particle Swarm Optimization
UAV	Unmanned Aerial Vehicle

ABSTRACT

Witcher, Paul Ryan. M.S.E.C.E., Purdue University, May 2017. Particle Swarm Optimization in the Dynamic Electronic Warfare Battlefield. Major Professor: Lauren Christopher.

This research improves the realism of an electronic warfare (EW) environment involving dynamic motion of assets and transmitters. Particle Swarm Optimization (PSO) continues to be used to place assets in such a manner where they can communicate with the largest number of highest priority transmitters. This new research accomplishes improvement in three areas. First, the previously stationary assets and transmitters are given a velocity component, allowing them to change positions over time. Because the assets now have a starting position and velocity, they require time to reach the PSO solution. In order to optimally assign each asset to move in the direction of a PSO solution location, a graph-based method is implemented. This encompasses the second area of research. The graph algorithm runs in $O(n^3)$ time and consumes less than 0.2% of the total measured computation time to find a solution. Transmitter location updates prompt a recalculation of the PSO, causing the assets to change their assignments and trajectories every second. The computation required to ensure accuracy with this behavior is less than 0.5% of the total computation time. The final area of research is the completion of algorithmic performance analysis. A scenario with 3 assets and 30 transmitters only requires an average of 147ms to update all relevant information in a single time interval of one second. Analysis conducted on the data collected in this process indicates that more than 95% of the time providing automatic updates is spent with PSO calculations. Recommendations on minimizing the impact of the PSO are also provided in this research.

1. INTRODUCTION

Particle Swarm Optimization (PSO) is an optimization method whose solution converges quickly and efficiently in scenarios with multiple constraints and objectives. The ease of creating and running a PSO, along with its speed performance compared to other optimization techniques, makes it an appealing and impressive tool. The PSO combines the characteristics of genetic algorithms and evolutionary computation approaches inspired by nature to solve complex scenarios in dynamic environments.

The technique of PSO was developed in 1995 by Professors Eberhart and Kennedy [1]. This optimization technique is motivated by the behavior of flocks of birds and schools of fish. Each member of the group is considered a particle, and a group of particles, a swarm. The swarm works towards a global optima, defined by various characteristics of the scenario. These characteristics are measured by the weight of each term in a fitness function, where the terms are representative of the parameters set by the user. The fitness function evaluates the suitability of the swarm's current state compared to a goal state that maximizes its value.

1.1 Overview and Problem Statement

This project continues the work conducted by Mr. Joshua Reynolds, Mr. Jonah Crespo, and the Crane Naval Surface Warfare Center (NSWC) [2] [3]. In this work, a PSO algorithm is implemented to identify the optimal locations at with to place assets such as Unmanned Aerial Vehicles (UAV). The computation of these locations occurs in a time span of less than one second, providing real-time updates to the dynamic electronic warfare (EW) battle space. Assets are placed in order to conduct EW operations with transmitters, each with a priority and particular frequency, located

on the ground. This placement also takes into account obstacles produced by 3D topography and allows dynamic and human interaction with the swarm [3] [4].

There are three objectives for this research. First, the behavior of the transmitters and assets needs to be extended from their currently defined static interaction with the battle space to one that is dynamic. The prior work assumed that the assets and PSO solution locations shared the same 3D coordinates. Therefore, a separate asset object needs to be added to the program in order to realistically simulate the assignment and movement of assets to PSO-defined locations in the EW battle space. Likewise, the transmitters will not always be stationary. For that reason, a velocity component will be added to each transmitter, allowing it to move across the battle space. Further, this research implements a method allowing the locations of the transmitters assets and PSO-defined locations to automatically update on a set time interval. The user is able to start and pause the update of these positions, as well as reset the battle space to an initial state.

The addition of dynamic asset and transmitter behavior prompts the need for the second objective. This objective addresses how the assets are assigned to individual locations of the PSO solution. A graph-based method is implemented to provide the optimal assignment of assets to the PSO solution locations. It runs in $O(n^3)$ time and has less than a 0.2% impact on the total computation.

With the addition of algorithms and logic necessary to implement the prior two objectives, the final objective will be to conduct code profiling. This process will identify which functions consume the most time in running the PSO application. Scenarios with different numbers of assets are tested. This research collects time measurements of the PSO calculations, asset navigation, Hungarian Algorithm, and redrawing of the GUI. These results are analyzed to pinpoint which functions are best suited for parallelization.

1.2 Literature Review

The concepts and theory involved in the use of PSO, dynamic asset allocation, and the Hungarian Algorithm have been implemented in other ways in the literature. These approaches demonstrate the flexibility of the aforementioned computational tools and show that the contributions in this research are unique and relevant.

1.2.1 Particle Swarm Optimization

PSO has been applied for asset allocation with microgrids and transmission systems. Mohan et al. [5] optimize a microgrid with PSO in order to distribute total load demand to microsources in a manner that maximizes the profit per unit of risk. They add the mechanism of stochastic weight tradeoff to the PSO, which balances the exploration of the population on a global and individual scale and diversifies the individual members of the swarm.

Rhein et al. [6] implement a PSO to optimize the maintenance and replacement of components in electrical transmission systems. As a result, the system maintenance is scheduled to maximize availability and reliability while minimizing the financial cost of said operations.

Al-Hmouz et al. use PSO to solve another type of problem [7]. Their implementation optimizes the distribution of information granularity in the analysis of time series.

A more prominent use of PSO for asset allocation can be found in financial scenarios. Liang and Qu [8] use PSO to select an investment strategy for a large scale portfolio. They modify the PSO to use Dynamic Multi-Swarms, where the entire population is composed of randomly grouped smaller swarms, to optimize the investment decisions. Similarly, Dang et al. implement a PSO to maximize wealth for dynamically allocated financial assets because of its low time of convergence [9]. Zhang and Zhang also modify a PSO to maximize the financial gain for asset allocation [10].

1.2.2 Dynamic Asset Allocation

Other approaches outside the realm of PSO have been used to optimize dynamic asset allocation. Parque et al. [11] use the Guided Genetic Relation Algorithm, a type of evolutionary computation technique, to adapt the allocation of assets in a financial portfolio as the market changes. Berksekas et al. uses a neural-dynamic programming framework to allocate defensive resources during a missile defense engagement [12].

Dynamic asset allocation in airborne scenarios has also been implemented with a variety of other approaches. Arslan et al. use dynamic programming to solve small allocation problems with airborne assets [13]. Their approach changes for problems of larger scale. These problems are solved with the use of hierarchical control and potential function methods. In both approaches, the authors aim to optimize the allocation of air assets and ammunition to ground targets.

Another implementation of dynamic airborne asset allocation is found in McDonnell et al. [14]. Evolutionary search is used to allocate assets for air strikes. Factors such as weapon effectiveness and risk are evaluated when optimizing the asset allocation. The strike force assets and targets are optimally coupled together, but in a non-spatial and non-continuous manner. This approach to asset assignment by McDonnell et al. is more limited in scope compared to the continuous assignment in three dimensions that occurs in this research.

Naval-based solutions for assigning military assets to targets in the scope of asset allocation are also found in the literature. Avvari et al. implements Voronoi tessellation for partitioning the search space for identification of high probability areas containing smugglers [15]. Flow maps are then created to direct the assets to the areas where counter-smuggling operations are most likely to succeed.

Another implementation that is more offensive in nature is accomplished by An et al. in [16]. The goal of the work by An et al. is to conduct counter piracy operations. These operations occur in two phases. Phase I allocated assets to intercept pirates and Phase II searches the regions not covered by the interdiction assets for other

threats. The Gauss-Seidel Algorithm is used in Phase I to guide assets to areas that have a high probability of pirate attacks. Phase II utilizes a partitioning algorithm coupled with an asymmetric assignment algorithm to search the regions not covered in the first phase for future assignments.

Raboin et al. apply dynamic asset allocation to a defensively-situated naval engagement [17]. They use market-based planning for task allocation of unmanned surface vessels to guard a particular area of interest. A genetic algorithm is then implemented to optimize the vessels' behavior, and a velocity vector is calculated to update their positions. Raboin et al. limit these assignments and subsequent interactions between vessels and task to the maritime realm, whereas the approach in this research for optimization of asset allocation involves interactions between ground and airborne units.

Cutler and Nguyen also dynamically allocate assets for a tactical air defense scenario in [18]. However, they use a rule-based model to allocate the resources. Oxenham and Cutler then build off of this work in [19] by adding obstacles to the scenario. A shortest path problem is then solved in a spherical geometry environment by combining a visibility graph with the use of Dijkstra's Algorithm.

A third type of military application is pursued in the work by Preece et al. in [20]. Their work utilizes a bidding protocol to assign sensor assets that operate in the realm of intelligence, surveillance, and reconnaissance to various mission tasks. However, the resource scheduling applied in the work by Preece et al. is simplified from that which would occur in a real world scenario.

1.2.3 The Hungarian Algorithm

As with the related works of PSO and dynamic asset allocation, the Hungarian Algorithm has been utilized in various implementations in the literature. The following sections will show that while the Hungarian Algorithm has been applied to the

problem of optimally assigning assets, the work pursued in this research is a unique contribution to what has been accomplished the current field of work.

The first interesting use of the Hungarian Algorithm is the modification implemented in [21]. Mills-Tettey et al. recognize that the Hungarian Algorithm will be applied to assignment problems where the edge weights or costs change over time. They present a version of the algorithm that is able to solve the assignment problem after its cost matrix has changed. It accomplishes this in $O(kn^2)$ time, where k is the number of changed rows or columns in the original cost matrix and n defines the size of a partition of the bipartite graph [21]. Future work could investigate the addition of the algorithm from Mills-Tettey et al. in this research, and if appropriate, implement it in place of the current $O(n^3)$ version.

Another use of the Hungarian Algorithm is implemented by Huang [22]. The author in [22] combines the Hungarian Algorithm with a genetic algorithm to find the shortest route possible for the traveling salesman problem. The scope in this research differs than that in Huang, as a solution to find an assignment of every asset to a solution has to be found, rather than a single shortest path.

The Hungarian Algorithm is also utilized to optimize situations with static locations and formations. Zhang et al. uses it to place robots in a static formation [23]. The total distance traveled by the group of robots is minimized in their work. Zhao et al. allocates sensor resources in a similar manner [24]. The sensors are stationary and are used to track and communicate with moving targets. To ensure reliable and accurate tracking, Zhao et al. optimize the assignment of sensors to targets. Similar use of the Hungarian Algorithm with sensor configuration can also be found in [25].

Several works in the literature also utilize the Hungarian Algorithm for dynamic situations. Liao et al. use the Hungarian Algorithm when deploying mobile sensors to create a wireless sensor network [26]. They focus on optimizing the target coverage and network connectivity with this algorithm. Optimization of the target coverage is achieved in their work by minimizing the distance the sensors travel. However, this is only applied to certain scenarios, with the Basic Algorithm and TV-Greedy Algorithm

being used in all other cases to minimize the total distance traveled. Meanwhile, a Steiner minimum tree is used to optimize the network connectivity.

Zhang and Wang utilize the Hungarian Algorithm when assigning tasks and targets to robots in [27]. Used in conjunction with the Genetic Algorithm, they are able to optimize the risk and time for the assignment of the robots. Targets assigned to the robots also contain a survival probability density that is accounted for when optimizing the assignment. Contrary to the independent behavior of each asset in this research, Zhang and Wang assume that all robots reach their targets simultaneously. Further, the assignment by Zhang and Wang for the robots is only updated two times during the simulation. Since the asset and transmitter positions of this research continuously update, the formulation and solution of the assignment problem is updated every second in the implementation of this research.

Turra et al. also optimize task assignment for unmanned vehicles with the Hungarian Algorithm [28]. Each unmanned vehicle has three possible tasks for each target; identification, verification, and attack. Timing constraints are utilized to space the execution of the tasks apart from one another. However, minimizing the distance traveled by the group is not the main objective for the optimization process by Turra et al. Further, they do not apply the real time update behavior found in this research to their work. Nonetheless, they do compare the efficiency of their use of the Hungarian Algorithm for optimization with the approach in [29]. Alighanbari et al. apply Mixed Integer Linear Programming (MILP) to coordinate the movement of UAVs. A more complex approach using the same formulation of the scenario as a MILP problem can be found in [30]. Here, Han et al. solve the MILP problem with the Lagrangian relaxation method and a dynamic list planning heuristic algorithm. Regardless, Turra et al. note that it is common for the computation required to solve MILP problems as they are scaled upwards increases to the point of being unusable for real time simulations [28].

1.3 Summary of Past Contributions

This research uses the prior work on PSO and asset allocation by Reynolds [2] and Crespo [3]. The application developed by them uses a population of particles to determine the optimal locations to place airborne assets in order to conduct EW operations with a group of ground-based transmitters. Crespo's improvements include adding a component of real-time human interaction with the swarm [3] [4] and the addition of 3D topography from the data collected by NASA's shuttle missions [3]. On account of the addition of 3D topography, Crespo enforces constraints on the available solution space to maintain the realism of the simulation.

Reynolds' and Crespo's work also provided this research with a GUI to run the PSO and display relevant information. This includes data on the PSO solution locations, the transmitter priorities, and the frequency bands in which the assets are assigned. A 2D representation of the transmitter locations, keep-away boundary, and asset locations, along with a Fitness Plot that measures the convergence of the PSO solution, are also included in this interface.

Three significant assumptions were made in the previous work. First, the transmitters were assumed to be static and unchanging in their position. Second, the PSO solution locations are assumed to be the same as the asset locations. With this assumption, the movement of the assets is modeled as jumping instantaneously to the PSO solution. The last assumption is that the same asset is assigned to the same PSO solution location.

1.4 Individual Contributions

This thesis describes the contributions made to solve the objectives outlined above. The first contribution involves transforming the static behavior of the transmitters and assets into dynamic behavior. This includes providing the user with the ability to start, pause, and reset the battle space simulation. Next, the second contribution is implementing a graph-based method, the Hungarian Algorithm [31] [32], alongside

the previously developed PSO from [2] and [3]. This implementation has a time complexity of $O(n^3)$ and consumes less than 0.2% of the total computation time spent updating the battle space information. The third contribution involves the collection of profiling data, its analysis, and recommendations on which parts of the project can be optimized and parallelized.

The main sections of this paper are defined as follows. Section 2 details the addition of transmitter movements and an automatic, user-controlled update of their positions. Section 3 explains the challenge of asset movement towards a PSO solution and the theory and methods used to solve this challenge. Section 4 measures the performance of this research's improvements and provides analysis of said measurements. Section 5 summarizes the contributions of this research and proposes improvements and changes in future work.

2. TRANSMITTERS

With the ever increasing presence and use of electronic warfare in today's battlefield, speed and accuracy are of the essence when observing and interacting with battlefield data. Due to testing and simulation constraints, the PSO was only run on static transmitter positions in the prior work [2] [3]. However, in a real world scenario, the transmitter positions would change frequently. Adding dynamic transmitter behavior to the application in this research allows the PSO to be tested in conditions similar to those experienced by the warfighter. By running the PSO with these dynamic transmitters, a solution is provided that defines the most up-to-date location in which each asset should reside.

The new transmitter behavior is implemented in three steps. First, a velocity component is added for each transmitter. Next, the transmitters use their velocity component to update their location after a constant time interval. Lastly, the project is modified to allow the user to observe automatic updates of the battle space and control when these updates occur.

2.1 Velocity Calculation

In the first step of simulating this desired realism, a velocity component was added to the structure in the C++ code for the transmitters. To model real transmitters, this can be input to the program by the user or passed to it by a programming interface. However, for testing purposes, this velocity is randomly generated for each transmitter and bound within a small range. The velocity is then added to each transmitter's current location, whose new location is then used by the PSO to calculate a new solution. The GUI displays these updates on the next time interval.

Inspiration was gained from the current implementation of the positions of the transmitters, which used the random number generator found in the Boost library to obtain a radius and theta value. These two values then are converted to rectangular coordinates and are set as the x and y transmitter velocities respectively. The x and y locations of each transmitter are updated with their assigned x and y velocity values.

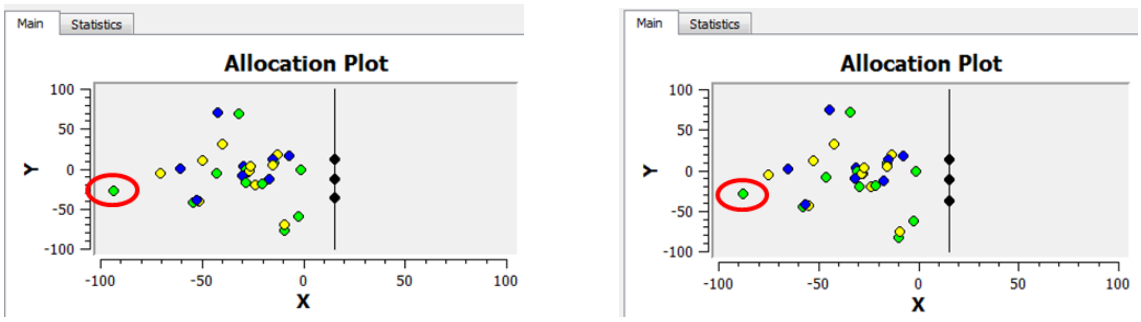
The z location of each transmitter is updated with data returned from a function that finds the elevation at the transmitter's new (x,y) position. In the simulations performed in this research, the ground units are bound to the surface of the terrain. If there are changes in the x or y coordinate for a particular transmitter, the elevation at that new (x,y) location will be retrieved, and the transmitter's z location will be updated, keeping it bound to the surface of the terrain.

2.2 Updating the Location of the Transmitters

For new transmitter position updates, a few modifications needed to be added to the code that handles the transmitter locations. These modifications ensure accurate and realistic behavior for the transmitters.

The first modification was adding a boundary check. This served two purposes. First, the boundary check made certain the transmitters remained in the viewable battle space while testing their behavior. Upon reaching this boundary, the transmitter in question simply reverses its direction by negating its velocity. The transmitter then uses this new velocity to update its location, ensuring that it does not leave the viewable battle space.

After this occurs, the velocities for each transmitter are added to its current position for the X and Y components. The second modification is implemented for the behavior of the transmitters in the z direction. A function uses elevation data to find a z value to assign to a transmitter's location at a particular (x,y) coordinate. The elevation data lookup only occurs for simulations that include a three-dimensional terrain file. When simulations are run without a terrain file loaded, the function



(a) Before encountering boundary.

(b) After encountering boundary.

Fig. 2.1. The green transmitter, circled in red, encounters the boundary of the battle space, and reverses direction. The time step between the two pictures is 4 seconds.

recognizes the lack of complementary elevation data and returns 0 as the value for the transmitter's z position.

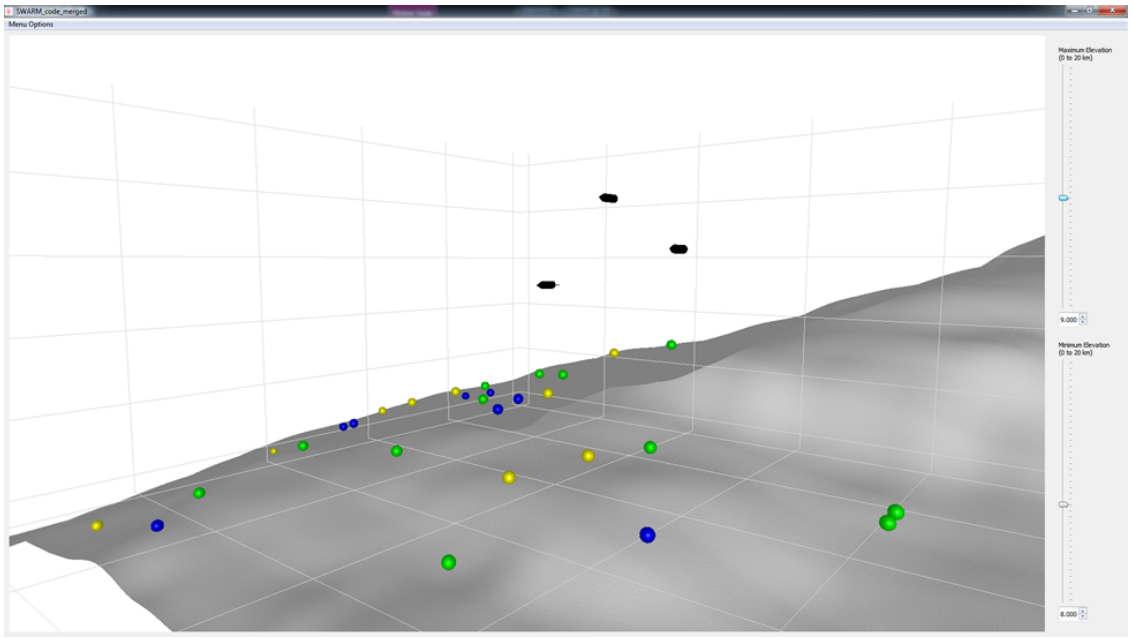


Fig. 2.2. The transmitters (yellow, green, and blue spheres) remain bound to the surface of the terrain. A section of Mt. Everest Terrain is shown above.

2.3 Updating the GUI on Set Intervals

As a result of the transmitter locations changing over time, the GUI needed to be updated in order to reflect the actual position of each transmitter. To implement these automatic updates, this research programs time intervals into the GUI.

```
QTimer *updateTimer = new QTimer(this);
connect(updateTimer, SIGNAL(timeout()), this, SLOT(updateGUI()));
updateTimer->start(1000);
```

Fig. 2.3. The timer code causes a GUI update every second.

The time intervals are modeled with the use of a timer in the project. The timer is connected to a timeout signal and a function that tracks the locations of the transmitters. This timer causes an update of the visual information to occur once every second. Because of the speed of the PSO, a new solution to the updated transmitter locations is calculated and displayed within this time interval.

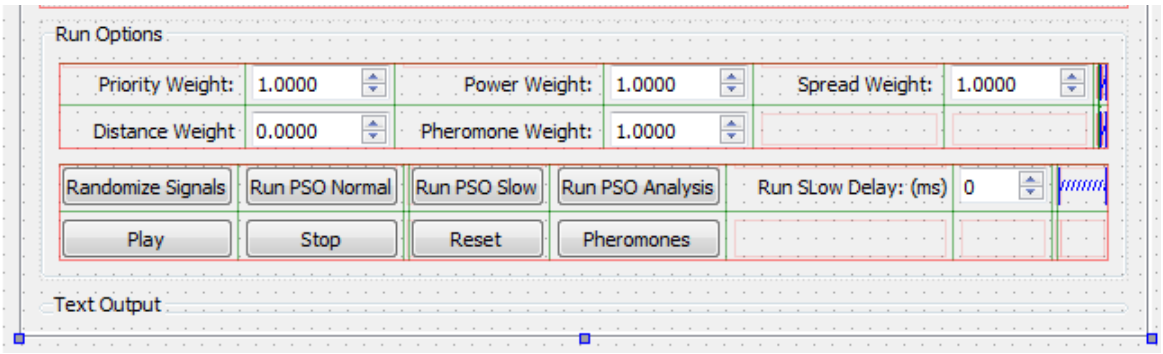


Fig. 2.4. The Play, Stop, and Reset buttons, above, allow the user to control the simulation of the EW battle space.

To begin this time lapse simulation, the user presses the Play button. The GUI updates the 2D and 3D positions of the transmitters every second. It also displays the new PSO solution during this interval. In order to pause the simulation, the user can click the stop button. Also, the simulation can be restarted from the paused state

by clicking the Play button. Finally, the user can return the GUI to its initial state by hitting the Reset button. This can be done without pausing the simulation first.

2.4 Summary

In conclusion, this research improved the static behavior of the transmitters in previous work [2] [3] to dynamic behavior that is more indicative of the situation the warfighter would face on the battlefield. This improvement was accomplished by adding a velocity component to the transmitters, updating the transmitter position with its velocity component, and modifying the GUI to display the updates of the battle space to the warfighter. These updates provide the warfighter with the most up-to-date locations, allowing them to place assets to accurately conduct electronic warfare operations with the transmitters.

3. ASSET MOVEMENT

Asset placement in the previous work [2] [3] instantaneously jumps to the PSO solution. In order to provide a more accurate simulation of the real-time, dynamic battlefield, these assets now have an initial position and speed, and move towards a PSO solution location. When set in motion, a matching PSO result location is assigned to each asset. The program determines the correct vector needed for each asset to reach its assigned PSO result location. The assets then use this information to update their location.

In order to calculate an assignment that minimizes the total distance traveled by all assets, this research utilizes graph theory. The Bipartite Matching Assignment is selected as a model for this challenge. The background of this approach in graph theory and the reasoning for its selection are described in the next section.

3.1 Bipartite Matching Assignment Problem

This research contains two sets of moving objects related to assets. The first set is the solution returned by the PSO, and the second set is composed of the current asset locations. These two sets both have the same number of elements.

The assets and PSO solution can be represented by vertices in a graph. Likewise, the distance between elements of the asset and PSO solution sets can be represented by weighted edges connecting the vertices of these two disjoint sets. Because every element of the set of assets is not a member of the set of elements in the PSO solution and vice versa, these two sets are disjoint and thus can be represented by a bipartite graph shown in Figure 3.1 on page 16.

Each asset needs to be assigned or matched to a unique location in the PSO solution. This matching needs to be completed in a manner that is computationally

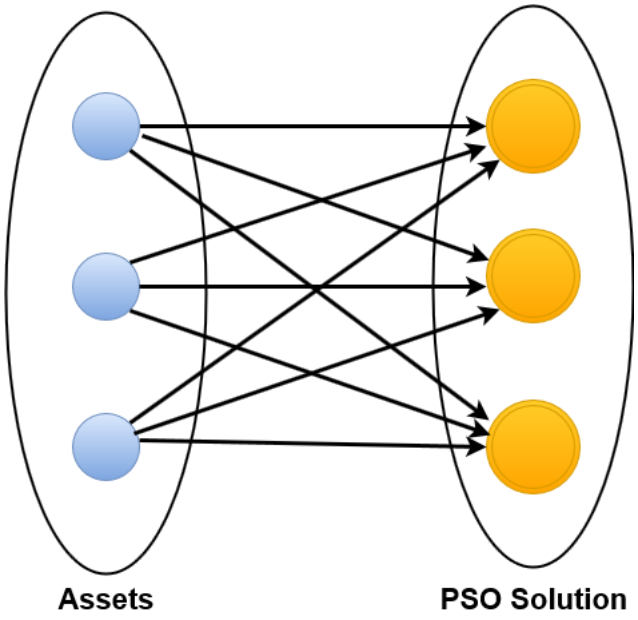


Fig. 3.1. The set of assets and the set of the elements of the PSO solution are disjoint and can be drawn as a bipartite graph

efficient. Further, the sum of the distances traveled by each asset should be minimized. This ensures that the set of assets can collectively begin comprehensive EW operations in the shortest timespan while maintaining those operations for the longest possible duration. In summary, this optimization goal is known as the Bipartite Matching Assignment Problem in the literature [33] [34]. The theory behind this formulation of the problem and possible solutions to it are discussed next.

3.1.1 Background and Theory

The theory behind the possible solutions to the Bipartite Matching Assignment Problem is described in three parts. First, a graph G with V vertices and E edges ($G = (V, E)$) is considered to be a bipartite graph if said graph contains the vertex partitions X and Y such that $V = X \cup Y$ (Figure 3.2 on page 17). Further, $X \cap Y = \emptyset$ and edges, $E \subseteq X \times Y$ [33].

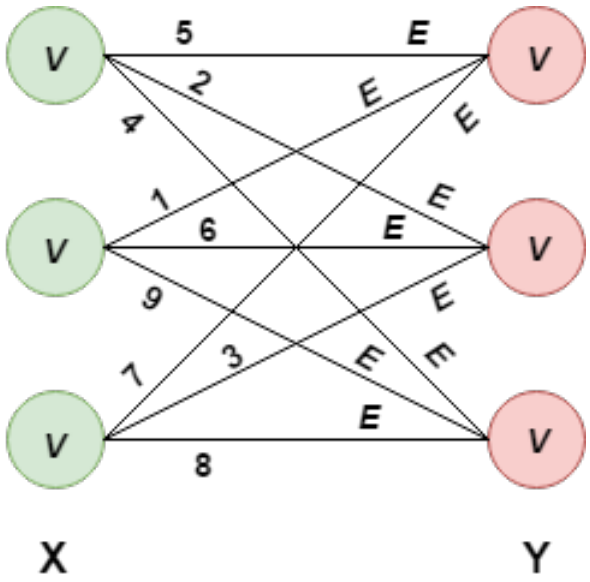


Fig. 3.2. A weighted bipartite graph, $(G = (V, E))$. Vertex partitions are labeled as X and Y.

Second, a matching edge graph M, such that $M \subseteq E$, is present if at most one of the edges in M is incident upon v, a set of vertices where $\forall v \in V$ [33]. In graph theory, a vertex and an edge are labeled as incident if the vertex is one of the endpoints of the edge in question [35]. Another way of saying this is that one edge in a matching set M cannot share an endpoint with another edge in the matching set M. Third, weights can be added onto the edges of the bipartite graph. When weights are present, a matching can be found that either maximizes or minimizes the sum of the weights of the edges.

Therefore, the assignment problem takes a complete, weighted graph as its input. A complete graph is one where every vertex is connected to every other vertex by a unique edge [35], as seen in Figure 3.2 on page 17. The matching of the optimal assignment, where the sum of the weights of the edges is optimized, is then returned. If such a matching set M can assign every element of X to every element of Y in such a way that respects the optimization of the weighted sum of the edges, then the matching is considered perfect, and the assignment is optimal [33].

3.1.2 Initial Attempt

First, the brute force approach for solving the assignment problem was attempted. The locations defined by the swarm solution were examined and the closest one to the asset in question was assigned to it. This location was then removed from the locations available for assignment. The process continued with the remaining assets until each one had a unique assignment.

At first glance, this solution appeared to handle the challenge of unique assignment between set of locations and the set of assets. However, upon further analysis, it was realized that this approach is not very efficient. While an assignment can be found quickly, there is no way of knowing if the assignment was the optimal one. Therefore, every possible assignment has to be checked. For n assets and n PSO solution locations, the number of possible assignment is $n!$ [32] [34]. In terms of time complexity, this approach is modeled as $O(n!)$.

Although this research conducts most of the tests and measurements described in the analysis section with 3 assets and 3 PSO solution locations, these constraints may change with actual, real-world use. Consequently, the factorial time complexity causes the brute force approach to significantly increase the computation in scenarios with 6 or more assets. The computational load from such an increase limits the range of battlefield situations able to be handled by this application. This increase also shifts the project away from its goal of providing real-time updates of EW operations to the warfighter. As a result, a more efficient approach was found, and is discussed in the next section.

3.2 Possible Solutions

3.2.1 Hungarian Algorithm

The first algorithm encountered when researching the assignment problem was the Hungarian algorithm. This algorithm takes a cost matrix as an input and manipulates

it to create elements with a value of zero in the cost matrix. Once a value of zero is present in every row and column, the indices (x, y) of these zeros determine the assignment of element x of set A to element y of set B.

This algorithm runs in O(n^3) time complexity and is straightforward to implement. Further, the algorithm allows the user the freedom to determine if the assignment provided by the Hungarian algorithm maximizes or minimizes the total cost. The ease of implementation, efficiency, and flexibility of this algorithm are the main reasons it was chosen to solve the assignment problem in this research. The theory behind this algorithm, as well as a proof of the time complexity, will be explained in Section 3.3.

3.2.2 Maximum-Flow Reduction Algorithm

The Maximum Flow Algorithm also initially appeared to be a possible solution to the assignment problem. Two applications of this algorithm, the Ford-Fulkerson Method and the Edmonds-Karp Algorithm [36], initially seemed to be applicable to this project. However, literature was found stating that the presence of weighted edges in the graph of the assignment problem would cause the Max-Flow Reduction to fail [33]. Even if such reduction was possible, the Ford-Fulkerson and Edmonds-Karp algorithms, due to their application of flow networks, required a source and sink node to be added to the beginning and end of the graph [33]. Implementation of such a requirement was deemed too complex and time consuming to run in concert with the PSO.

3.3 Selected Approach

The Hungarian algorithm [31], also known as the Kuhn-Munkres algorithm [33], is named to honor two Hungarian mathematicians, Kőnig and Egerváry, whose work is the basis of the algorithm. Kuhn published his paper, The Hungarian Method for the Assignment Problem, in 1955. In it he details how the algorithm runs in finite time

and explains the steps of the process to follow to solve the assignment problem [31]. A summary and analysis of Kuhn's work is presented in the following section.

3.3.1 Kőnig's Contributions

First, Kuhn explains how the work by Kőnig is used as the theoretical basis for the Hungarian algorithm [31]. Kőnig defines the assignment problem as assigning jobs to individuals. This problem is simplified from the general definition of the assignment problem by only denoting a zero or one for each intersection of a job and individual. In this case, a one denotes that an individual is qualified for a job and a zero denotes that the individual in question is not qualified [31]. To represent this relationship, a qualification matrix is created. This matrix is a simplified version of a cost matrix in the general assignment problem.

One aim of Kuhn's paper is to find the largest number of ones that can be selected from the qualification matrix, with each value of one having a unique column and row. An assignment is a selection of these ones, and is considered complete if it cannot be expanded to include more matchings between individuals and jobs. This assignment would be labeled incomplete if an unassigned individual is able to be matched to an unassigned job. Improvements are made on incomplete assignments by what is termed a *transfer*. If the ones in the current assignment can be shifted to allow an individual to be matched to an unassigned job, such a *transfer* is possible.

The following paragraphs describe the theorems developed by Kuhn as the basis of the Hungarian Method.

> **"Theorem 1** An individual, job, or both the individual and job are *essential* if every *transfer* on the given assignment results in a complete assignment." [31]

> **"Theorem 2** There exists a complete assignment after every possible *transfer*." [31]

At this point, Kuhn adds a parallel interpretation of the assignment problem. The concept of a *budget* is introduced, where each individual is allotted a value for the

job he or she is qualified for. As before with the qualification matrix, these values are either a one for a qualified individual or a zero if the individual is not qualified. Kuhn defines a *budget* as *adequate* if it provides a value of one to the individual, job, or both that is part of a particular match of qualified individual to a job. He summarizes this in the following theorem.

> **"Theorem 3** The allotment of a *budget* that is *adequate* cannot be less than the number of qualified individuals that can be assigned to jobs." [31]

From the proof of Theorem 3 and the language of Theorems 1 and 2, Kuhn proposes the next theorem.

> **"Theorem 4** There exists an *adequate budget* and assignment where the total allotment of said *budget* equals the number of jobs assigned to qualified individuals." [31]

With this, Kuhn arrives at the conclusion that the largest number of jobs that can be assigned to qualified individuals is equal to the smallest total allotment of any *adequate budget*. Any assignment is optimal if and only if it is complete after every possible *transfer* [31].

3.3.2 Egerváry's Contributions

Now that the assignment problem has been shown to have an optimal solution when reduced to a simplified one and zero state, Kuhn uses Egerváry's work to demonstrate how a general assignment problem can be reduced to such a state while obtaining an optimal solution in finite time.

Before the continued development of theorems, Kuhn defines a *rating matrix* that provides a numerical *rating* for a particular individual's performance (rows of the matrix) for a particular task (columns of the matrix) [31]. The summation of the ratings from an assignment is considered a *rating sum*, which will be maximized in the optimal case [32].

The first theorem Kuhn develops in relation to the graph theory and mathematical definitions produced by Egerváry involves *adequate budgets* and reaches a similar conclusion as Theorem 3.

"Theorem 5 The total allotment of any *adequate budget* is not less than the *rating sum* of any assignment." [31]

From this, Kuhn states that any time in which an assignment and *adequate budget* together form a total allotment equal to the *rating sum*, they compose a solution to the assignment and *budget* problems. Therefore, Theorem 6 follows:

"Theorem 6 If all n individuals can be assigned to jobs for which they are qualified in the Simple Assignment Problem (all zeros and ones) associated with an *adequate budget*, then the assignment and the budget solve the given General Assignment Problem and the *rating sum* equals the total allotment." [31]

Kuhn then discusses how to improve the *budget* when individuals have not been assigned to jobs they are qualified for in the Simple Assignment problem. The total allotment is reduced by $n - r$, where n is the number of jobs and r the number of *essential* individuals. Then it is increased by s, the number of *essential* jobs. Since $r + s = m$, with m being the largest number of qualified individuals assigned to jobs and $m < n$, the net reduction for the total allotment is $n - m$, whose difference is always nonnegative [31]. Therefore, Kuhn proposes the last theorem.

"Theorem 7 If at most $m < n$ individuals can be assigned to jobs for which they are qualified in the Simple Assignment Problem associated with an *adequate budget*, then the total allotment of the budget can be decreased by a positive integral amount." [31]

Accordingly, either theorem will be applied to the assignment problem. If an *adequate budget* is optimal, then Theorem 6 is applied to the problem. If said *budget* can be decreased, Theorem 7 applies and the *budget* is decreased. Because Theorem

7 can only be applied a finite number of times before Theorem 6 becomes applicable, Kuhn concludes the following about the General Assignment Problem.

"The largest possible rating sum for any assignment is equal to the smallest total allotment of any adequate budget. It can be found by solving a finite sequence of associated Simple Assignment problems." [31] Consequently, Kuhn can now apply this combination of approaches to the Assignment Problem into an algorithm.

3.3.3 The Hungarian Method

First, a *rating matrix*, R, is examined. A maximum of every row, a_i, is found. Likewise, a maximum of every column, b_j, is found. All row maximums are summed into a single summation, a. In the same way, all column maximums are summed into a single summation labeled b.

If $a > b$, then $v_j = b_j$ for $j = 1, 2, ..., n$ and $u_i = 0$ equals zero for $i = 1, 2, ..., n$. However, if $a \leq b$, $u_i = a_i$ for $i = 1, 2, ..., n$ and $v_j = 0$ for $j = 1, 2, ..., n$. u_i and v_j are nonnegative integers that, when added together, equal the value of the rating matrix at (i, j).

Second, a new matrix, R', is created with the u_i and v_j elements. Each position, (i, j) of the matrix is defined as $u_i + v_j$. With these values calculated, the matrix R' is compared to the initial *rating matrix* R. Any index of R that has the same value as the equivalent index in R' is marked at that index with the value of a one in a new matrix, Q. Any values that are different between matrices R and R' are marked with a value of zero.

Third, the matrix Q is examined for a set of independent ones. A set of ones is considered independent if they do not share the same row and column. This set of ones is then marked with asterisks (1^*).

Fourth, the matrix Q is searched for (1^*). If an asterisk is found, its column is searched for a one. If no values of one are found, then the row the asterisk resides in is *essential*. Likewise, a column is *essential* if it contains an asterisk in an *inessential*

row. From the definition of an *essential* row, an *inessential* row is one that has a one with an asterisk along with other ones in different rows of the same column.

Fifth, the current assignment is examined for possible *transfers*. If no *transfers* are available at this point, skip to the next step. If transfers that free a column containing an unassigned one are possible, they are carried out. This occurs until a complete assignment is found. The last row involved in the *transfer* is considered *essential*.

If no *transfers* were possible (before the complete assignment), all assigned columns are *essential*. Recall that a column is assigned if it contains a one with an asterisk. Next, a matrix D is created by subtracting elements of matrix R from elements of R'. The differences in the *inessential* rows and columns of matrix D are then examined and the minimum is found and labeled as d. If this is not possible, then a solution has been found. Otherwise, the next step is taken.

With a minimum d found, one of two cases occurs. If $u_i > 0, \forall i$ *inessential* rows, find $m = \min_{\forall i}(d, u_i)$. Then update u_i by subtracting m from it $\forall i$ in *inessential* rows. Update v_j in a similar manner $\forall j$ in *essential* columns.

However, if $u_i = 0$ in one of the *inessential* rows, find $m = min_{\forall j}(d, v_j)$. After that, update all $u_i = u_i + m, \forall i$ in *essential* rows. Then update all $v_j = v_j - m, \forall j$ *inessential* columns.

Once one of the two cases above is carried out, the algorithm returns back to the second step and progresses through the subsequent steps until a solution is found.

3.3.4 Munkres' Improvements

James Munkres published a paper two years after Kuhn detailing improvements and an analysis of the time complexity of the Hungarian Algorithm [32]. His improvements are summarized in the following paragraphs, with the time complexity analysis explained in the next section.

Like Kuhn's approach [31], Munkres begins with a rating matrix [32]. He then conducts a few preliminary operations. First, the smallest element of a row is subtracted from every element in that row. This is repeated for every row in the rating matrix. Next, the smallest element of each column is subtracted from every element residing it that column. Then, the zeros are observed. All independent zeros are starred. Zeros are independent if they do not share a row or column with another starred zero. Every column containing a starred zero is covered. To cover a column, one simply denotes a line running through the column in question. With this complete, the improved algorithm is described in the following three steps [32].

1. Find and prime all zeros that are not covered. Then look at each primed zero. If a starred zero does not reside in the same row as a primed zero, go to the following step. If it does, cover the row with the starred and primed zeros, and uncover the column in which the starred zero resides. Repeat this process until all zeros have been covered, and then go to Step 3.

2. Starting with every zero that is uncovered and primed, look in the column the uncovered and primed zero resides for a starred zero. If one exists, look in the row of the starred zero for a primed zero. This sequence continues until a zero cannot be found to satisfy these restrictions. The last zero found should be primed.

   Remove the star on every starred zero and star every primed zero of the sequence. This will result in a set of independent starred zeros that is larger in size by one element than the previous set of starred zeros. Unprime every zero that still has a prime symbol, uncover every row, and cover every column that contains a starred zero. At this point, if every column is covered, then the starred zeros form a solution. If this is not the case, return to Step 1.

3. Let h be the smallest uncovered element of the matrix. Add h to every element of the covered rows. Subtract h from every element of the uncovered columns. Return to Step 1.

Time Complexity

Thus, with Kuhn's algorithm improved, Munkres calculates the time complexity of the new method [32]. Beginning with a matrix of m independent starred zeros, Munkres considers the maximum number of operations necessary to obtain a matrix with $m+1$ independent starred zeros. The following analysis follows Munkres.

The preliminary operations conducted on the initial matrix will need a most $5n+4$ (where n is defined as a dimension of an $n \times n$ matrix) operations to obtain one starred zero.

Next, Step 1 with a matrix of m starred zeros need $n+4$ operations in the worst case to cover one horizontal line of the matrix.

For Step 3, $2n+m$ operations are necessary to resolve the worst case scenario of finding the smallest uncovered element, h, and adding and subtracting it to the necessary elements. Time n is taken for scanning a column, another n amount of time to subtract h from every element of a column, and m amount of time for adding h to each element of a covered row. Computing the total amount of time for Step 3 to be completed results in $(2n+m) \times (m-1)$ total time needed.

Once the algorithm returns to Step 1 from Step 3, it will execute $n+4$ operations until m covered horizontal lines are present on the matrix. In the worst case, this takes $(m-1) \times (n+4)$ time.

When Step 1 leads to Step 2 in the case of each uncovered row containing an uncovered zero, $n+1$ operations at the most will be needed to transition to this step. Once inside of Step 2, $2m$ time is needed to scan a line, $2m+1$ time to erase a prime or an asterisk, $m+1$ time to star a zero, and $2m+1$ time to cover or uncover a line. Adding these operations together results in $7m+3+n+1$ time.

The time for executing Steps 1-3 will be summed from $m = 1$ to $m = n-1$. Because the preliminary step is not repeated outside its initial execution, it is added on to the result of the summation. The math for this is shown below.

Calculation of Theoretical Run Time Complexity

1. The initial summation combining the run time for each step described above.

$$\sum_{m=1}^{n-1} \{(n+4) + [(2n+m) \times (m-1)] + [(m-1) \times (n+4)] + (n+1) + (7m+3)\} \tag{3.1}$$

2. After simplifying, the resultant summation is:

$$\sum_{m=1}^{n-1} (4 - n + m^2 + 3nm + 10m) \tag{3.2}$$

3. Separating the summation into smaller ones and evaluating each one gives:

$$(4-n)\sum_{m=1}^{n-1} 1 + \sum_{m=1}^{n-1} m^2 + (3n+10)\sum_{m=1}^{n-1} m \tag{3.3}$$

$$(4-n)\sum_{m=1}^{n-1} 1 = 5n - n^2 - 4 \tag{3.4}$$

$$\sum_{m=1}^{n-1} m^2 = \frac{2n^3 - 3n^2 + n}{6} \tag{3.5}$$

$$(3n+10)\sum_{m=1}^{n-1} m = \frac{9n^3 + 21n^2 - 30n}{6} \tag{3.6}$$

4. And combining gives:

$$(5n - n^2 - 4) + \frac{2n^3 - 3n^2 + n}{6} + \frac{9n^3 + 21n^2 - 30n}{6} = \frac{11n^3 + 12n^2 + n - 24}{6} \tag{3.7}$$

5. Once the preliminary time cost $(5n + 4)$ is added and the resultant sum is simplified, the following is the result.

$$\frac{11n^3 + 12n^2 + n - 24}{6} + \frac{30n + 24}{6} = \frac{11n^3 + 12n^2 + 31n}{6} \tag{3.8}$$

6. When considered at asymptotic bounds, it is clear that the time complexity is $O(n^3)$.

3.4 Implementation of the Hungarian Algorithm

After analyzing the possible ways to solve the assignment problem, the Hungarian Algorithm was chosen as the most efficient and straightforward method. With this in mind, prior implementations were investigated.

The C++ dlib library provides a multitude of matrix manipulation techniques, optimization algorithms, and machine learning implementations. Within this library is a max_cost_assignment function that runs the $O(n^3)$ implementation of the Hungarian algorithm on an input of a cost matrix. This function then returns the solution that maximizes the cost of the assignments.

3.4.1 Dlib's Version of the Hungarian Algorithm

The version of the Hungarian Algorithm implemented in this dlib function is outlined in the tutorial initially followed to create the algorithm header files from scratch. A few definitions need to be defined first.

A graph $G = (V, E)$ of V vertices connected by E edges is given. This graph can be partitioned into two sets, X and Y, where $X \cup Y$ contains all of the vertices of the graph. Additionally, the intersection of these two sets must be empty ($X \cap Y = \emptyset$). Edges can then be drawn between both sets. Each edge is then weighted with a value that represents a cost between two vertices. When a vertex is not connected by an edge, this vertex is said to be exposed.

Graph G is said to be labeled when the vertices of an edge are each given a weight. If the sum of these weights is greater than or equal to the weight of the edge, then the labeling is feasible.

A path tracing the edges between sets X and Y is considered alternating if each edge has a vertex from a different set. More specifically, each edge of this path begins in set X and ends in set Y and vice versa. If the first and last vertices in an alternating path are exposed, the path is considered an augmenting path.

An equality subgraph is said to span the given, labeled graph if it is composed of vertices and edges where the sum of the labels of the vertices of each edge equal the edges weight. When this occurs, the labeling is perfectly feasible.

If the equality subgraph contains a perfect matching, that is if the matching is complete, then the matching of the subgraph is the maximum sum of weights. Therefore, this matching is optimal and is the solution to the assignment problem described by the initial graph.

Before outlining the steps needed, two definitions from [37] need to be provided first. v's neighborhood is defined by the vertices that share an vertex with v ($v \in V$), $J_G(v) = \{u \mid (v, u) \in E\}$. Also, $S \subseteq V$, S's neighborhood is all vertices that share an edge with a vertex in S, $J_G(S) = \bigcup_{v \in S} J_G(v)$.

The following approach is composed of four steps.

1. Find some initial feasible vertex labeling and some initial matching.

2. If the matching M is perfect, then it is optimal, and the problem has been solved. Otherwise, there is an exposed $x \in X$ that exists. Set S contains the member x and set T is empty. x will be the root of the alternating path built in the next step.

3. If $J_{G_l}(S) \neq T$, go to step 4. Otherwise find α, the minimum of the difference of the sum of the labels of x and y the weight of the edge xy. A new labeling is constructed by subtracting α from every label $v \in S$, adding α to every $v \in T$, and leaving all other labels in either set the same. G_l is replaced with $G_l{}'$.

4. Find some vertex $y \in T \setminus J_{G_l}(S)$. If y is exposed, then an alternating path from x to y exists. The path is augmenting, and Step 2 is now executed. However, if y is matched in M with some vertex z, add (z, y) to the alternating path and set $S = S \cup \{z\}$ and $T = T \cup \{y\}$. Return to Step 3.

This completes the theory section. The next section provides results from the implemented research.

4. ANALYSIS

4.1 Validation of Hungarian Algorithm

Integrating the dlib function, max_cost_assignment, into the current PSO project involved two steps. First, a standalone testing environment was created in Microsoft Visual Studio. This testing environment was used to understand the function, how to call it properly, and how to use its output assign assets to swarm-defined locations. This isolation allows all performance and run time analysis to be focused on the logic particular to implementing the Hungarian Algorithm. The next step involves integrating this implementation of the algorithm into the project. There the analysis of the interaction between the Hungarian Algorithm and the PSO is conducted. Because this interaction now utilizes real location data, the speed and accuracy of the program can be measured in a setting similar to what the warfighter would experience on the battlefield.

4.1.1 Standalone Environment

The source files from version 19.1 of the dlib were used in the validation and testing of the behavior and performance of the Hungarian Algorithm. The dlib source file is combined with our research files to interface to the PSO. The new files contain the necessary interfaces to solve the assignment problem with a function that returns a maximum cost assignment. The limitations of this assignment function, as well as the manner in which this research solved them, are described below.

Input Limitation

The first limitation of the dlib function was the fact that it could only take in a cost matrix of **integer** type. The file included in the dlib states the reason for this is that the data types used must result in reliable outcomes when compared with the == operator.

Another limitation was the fact that the dlib function needed a matrix data structure as input. This restriction turned out to be inconsequential as the dlib source files include a definition for the matrix data type.

Maximum Cost to Minimum Cost Conversion

The most important limitation is the fact that the dlib function only **maximizes** the cost assignments from the cost matrix provided. In this research, the distances between the assets and their ideal locations, as defined by the PSO, compose the cost matrix. For our application, a function is needed which minimizes, not maximizes, the distance the assets have to travel. Therefore, the dlib function needed to be converted to return an assignment that **minimizes** the distance.

Because the input of the dlib function is a matrix, a minimum cost assignment could be returned by a maximum cost assignment function by manipulating the input matrix. At first, this problem appeared to have a straightforward solution of simply negating every element of the cost matrix. However, the Hungarian Algorithm requires positive integer values in order to find an accurate assignment. Upon further analysis and testing, the cost matrix is manipulated as follows:

1. Find the maximum valued element of the matrix.
2. Divide every element of the cost matrix by this maximum value.
3. Find the reciprocal of every element by inverting it.
4. Multiply each element by 10. Take the ceiling of the result.

5. Input the new cost matrix into the dlib function.

As seen in the example below, the assignment returned that maximizes the cost of the manipulated matrix minimizes the cost of the original cost matrix.

Example

1. A matrix is created from the distance between every asset and every PSO location. The rows of the matrix are the assets, and the columns are the PSO solution locations. These two numbers are equivalent, resulting in a square matrix.

   Following the first step above, the maximum value of the matrix, 20, is identified. The assignment elements that result in an assignment with minimum cost are boxed in below.

$$\begin{pmatrix} \boxed{4} & 12 & 10 & 11 \\ 12 & \boxed{6} & 16 & 15 \\ 16 & 20 & 18 & \boxed{16} \\ 13 & 16 & \boxed{15} & 14 \end{pmatrix}$$

2. Each element is divided by the largest element, 20. The simplified fractions are shown below.

$$\begin{pmatrix} \frac{1}{5} & \frac{3}{5} & \frac{1}{2} & \frac{11}{20} \\ \frac{3}{5} & \frac{3}{10} & \frac{4}{5} & \frac{3}{4} \\ \frac{4}{5} & 1 & \frac{9}{10} & \frac{4}{5} \\ \frac{13}{20} & \frac{4}{5} & \frac{3}{4} & \frac{7}{10} \end{pmatrix}$$

3. Every element of the matrix is inverted.

$$\begin{pmatrix} 5 & \frac{5}{3} & 2 & \frac{20}{11} \\ \frac{5}{3} & \frac{10}{3} & \frac{5}{4} & \frac{4}{3} \\ \frac{5}{4} & 1 & \frac{10}{9} & \frac{5}{4} \\ \frac{20}{13} & \frac{5}{4} & \frac{4}{3} & \frac{10}{7} \end{pmatrix}$$

4. Multiplication by 10 of each element takes place, followed immediately by the ceiling operation on the result.

   The multiplication increases the number of distinct elements in the matrix. Executing the ceiling function on these elements converts them to integers, allowing them to be used with the max_cost_assignment function from the C++ library, dlib.

$$\begin{pmatrix} 50 & 17 & 20 & 19 \\ 17 & 34 & 13 & 14 \\ 13 & 10 & 12 & 13 \\ 16 & 13 & 14 & 15 \end{pmatrix}$$

5. Input the new cost matrix (pictured on the left) into the dlib maximum cost assignment function. The returned assignment is $[0, 1, 3, 2]$, marked by the boxes around each element in that the transformed matrix.

$$\begin{pmatrix} \boxed{50} & 17 & 20 & 19 \\ 17 & \boxed{34} & 13 & 14 \\ 13 & 10 & 12 & \boxed{13} \\ 16 & 13 & \boxed{14} & 15 \end{pmatrix}$$

With the process complete, the assignment found for the original matrix can be compared with the assignment found for the transformed matrix. The transformed matrix is shown on the left with its assignment boxed in. On the right, the original matrix is shown with elements from the same assignment boxed in.

$$\begin{pmatrix} \boxed{50} & 17 & 20 & 19 \\ 17 & \boxed{34} & 13 & 14 \\ 13 & 10 & 12 & \boxed{13} \\ 16 & 13 & \boxed{14} & 15 \end{pmatrix} \qquad \begin{pmatrix} \boxed{4} & 12 & 10 & 11 \\ 12 & \boxed{6} & 16 & 15 \\ 16 & 20 & 18 & \boxed{16} \\ 13 & 16 & \boxed{15} & 14 \end{pmatrix}$$

When these boxed elements are added together for the transformed matrix, a cost of 111 is obtained ($50 + 34 + 13 + 14 = 111$). Likewise, when this addition occurs for the same assignment in the original matrix, the cost of 41 is obtained ($4 + 6 + 16 + 15 = 41$).

Comparing these costs with the other possible costs that result from different assignments within their matrix, it is clear that the sum found for the transformed matrix is the maximum cost assignment for that matrix. The same holds true for the original matrix.

It should be noted that this is an 'engineering' solution for converting the input matrix so that a minimum cost assignment is returned by the max_cost_assignment dlib function. This conversion, while not proven mathematically, has returned minimum cost assignments that have been verified by hand as being accurate. Therefore

this approximation, in all cases tested in this research, provides an assignment that both maximizes the total cost of the transformed matrix and minimizes the total cost of the original matrix.

4.1.2 Time Complexity

Theoretical Calculation

The time complexity of this process can be calculated theoretically. Measurements of run time will then be used to verify the theoretical complexity.

First, the input of the program, a vector, is passed by reference and thus takes constant time. This vector contains the distances between each asset and its end location, which can be labeled as n assets and n locations. The number of assets and locations are the same, so the number of elements contained within the vector is $n \times n$, or n^2. The vector is then converted to an array, which allows for the maximum valued element to be found quickly. In the worst case scenario, this takes $\mathrm{O}(n^2)$ time since there are n^2 total elements to search.

Next, the arithmetic operations of division and multiplication are conducted on each element of the array. Because these operations take constant time when combined, and they are performed on every element, the total time complexity for this step is $O(n^2)$.

After that, the ceiling function is performed on each element. Since this function is linear in regards to the input, the time complexity here is $\mathrm{O}(n^2)$.

Finally, the Hungarian Algorithm is run. As explained above, the run time for the dlib implementation of this optimization algorithm is $\mathrm{O}(n^3)$. When combining all the time complexities from each step above, the resultant time complexity at asymptotic values for the entire process is $\mathrm{O}(n^3)$.

$$n^2 + n^2 + n^2 + n^3 = 3n^2 + n^3 = O(n^3)$$

Elapsed Time Data

With the testing environment isolated from the rest of the PSO project, the true impact and efficiency of this implementation of the Hungarian Algorithm can be measured. To verify that the time complexity of the Hungarian Algorithm is still $O(n^3)$, the code was run and timed with different cost matrix dimensions (number of assets). The minimum matrix dimension tested in this experiment, three, is the most commonly used dimension. The maximum, 352, is the point at which no more memory can be allocated to the cost matrix for the computations to be completed. The other dimension values tested fall between these two extremes and were chosen to best explore the relationship between input size and elapsed time.

Ten trials were executed consecutively for each matrix dimension (number of assets), and the elapsed time was observed. The measurement of elapsed time for the Hungarian Algorithm was limited to the function that transformed the input matrix and executed the dlib function. The average of the ten trials for each dimension was then obtained and can be observed in Table 4.1 on page 37. These results, and the analysis of them, confirm the theoretical run time calculation and can be viewed in the following tables and charts.

In order to assess the time complexity of the real world results, four different trendlines were fit onto Figure 4.1 (page 38). This research tested four types of trendlines; one linear and three polynomial, with the polynomial trendlines being of order two, three, and four. The equation and coefficient of determination (R-squared value) for each of the trendlines was then recorded and analyzed. The R-squared value measures how well the data fits the trendline calculated [38]. These values range from zero to one, where an R-squared value of zero demonstrates that the trendline does not fit the measured data very well, whereas a value of one means it fits the data perfectly. As seen in Table 4.2 on page 39, the R-squared values for the polynomial trendlines are all greater than .99, with the second order having the highest R-squared value.

Table 4.1.
Trial Times (ms) for each Dimension

Asset	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Trial 6**	**Trial 7**	**Trial 8**	**Trial 9**	**Trial 10**	**Average Time**
3	0.0107	0.0078	0.0078	0.0119	0.0086	0.0115	0.0082	0.0082	0.009	0.0082	0.0092
10	0.0238	0.0197	0.0201	0.0247	0.0218	0.0193	0.0251	0.0193	0.0913	0.021	0.0286
25	0.106	0.0567	0.0555	0.0707	0.0625	0.0903	0.0629	0.097	0.0703	0.0551	0.0727
50	0.3271	0.2137	0.2174	0.2289	0.2182	0.1944	0.2252	0.2116	0.3287	0.3189	0.2484
100	0.9229	0.685	0.6674	0.9365	0.7738	0.7894	1.1789	0.9024	0.8272	1.3848	0.9068
150	2.3891	1.6289	1.6663	2.3119	1.9178	3.0092	2.2794	1.6716	1.7583	2.1557	2.0788
200	3.6539	3.2545	3.6699	3.0852	3.7747	3.228	3.365	3.4682	3.5837	3.4986	3.4582
250	4.996	5.261	6.1379	5.8174	5.9764	5.5906	6.5176	6.6163	5.0749	6.0673	5.8055
300	9.1450	8.2057	11.8871	7.4023	11.8148	8.6840	7.5910	10.4378	8.6931	8.5033	9.2364
350	12.5195	11.9825	13.4445	15.5242	12.2594	10.1752	11.4812	11.8432	14.5002	11.9032	12.5633
352	12.2109	14.6346	12.9009	14.2425	12.3404	12.0864	12.0975	11.378	13.3225	12.5643	12.7778

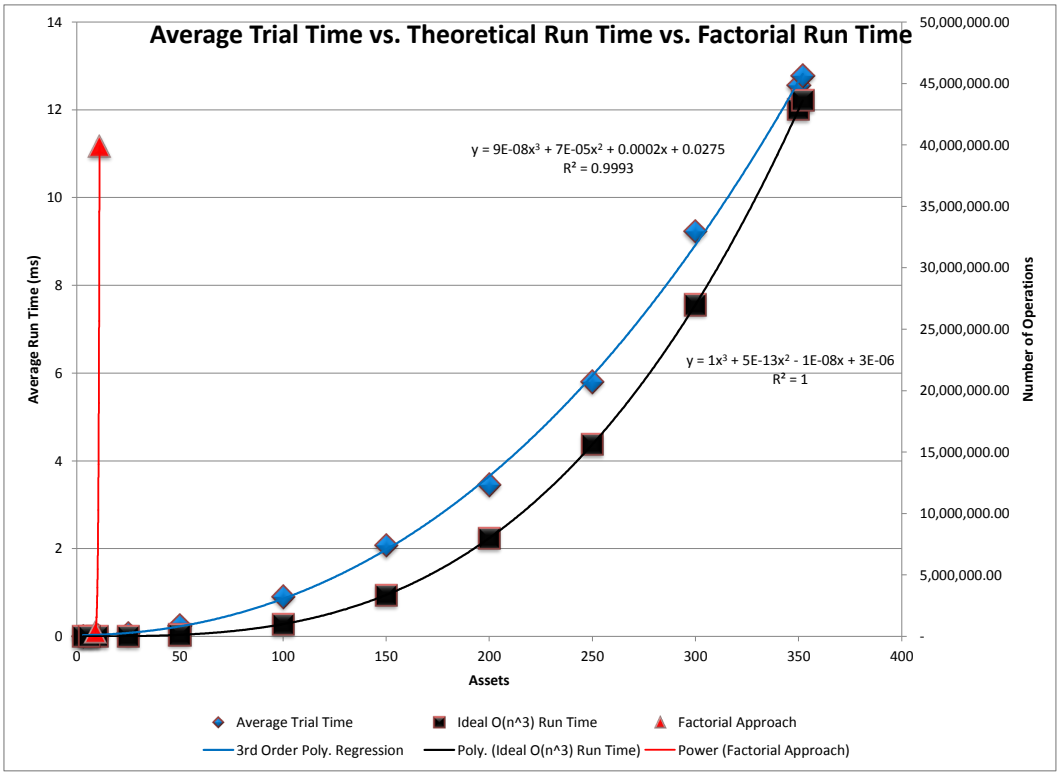


Fig. 4.1. A comparison of a measured run time trendline with two theoretical run time trendlines, $O(n^3)$ and $O(n!)$.

However, the R-squared value should not be the only tool used to verify the type of best fit line that matches the data. Regression residuals are used to confirm that the R-squared value, and in turn the trendline being used, are the best match for a dataset [39]. These residuals measure the difference between the predicted value and the measured value [40]. This measurement is then used to evaluate how predictable the model is. When graphed, the residuals should be independent and normally distributed with zero mean [41]. If this were not the case, the error between

Table 4.2.
Regression Line Equations and R-squared Values

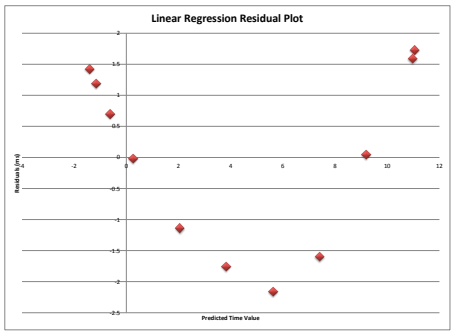
Regression Type	**R-squared Value**	**Equation**
linear	0.9173	$y = 0.0357x - 1.5213$
2nd order polynomial	0.9999	$y = 0.0001x^2 - 0.0059x + .1381$
3rd order polynomial	0.9993	$y = (9 \times 10^{-8})x^3 + (7 \times 10^{-5})x^2 + .0002x + .0275$
4th order polynomial	0.9995	$y = (-8 \times 10^{-10})x^4 + (7 \times 10^{-7})x^3 - (6 \times 10^{-5})x^2 + .0093x - .075$

the observed data and predicted data could be calculated, which indicates that the trendline does not accurately account for all the data points [39].

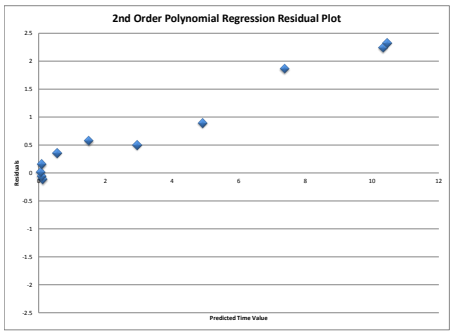
Although the linear best fit trendline has a high R-squared value, it is clear that it is not normally distributed (Figure 4.2(a) on page 40). One can easily predict that all residuals for time measurements between zero and nine will be negative, whereas all residuals for times greater than nine will be positive. A parabolic curve in the residual scatter plot is indicative of the need for a polynomial best fit line [41].

Residual plot analysis can also determine the best polynomial trendline to use for the data. When a polynomial of second order is used, a pattern is found in the residuals (Figure 4.2(b) on page 40). None of them of them are negative after a time measurement of 0.5 ms, and as the time measurement increases, the distance between the residual and zero does as well. This indicates the error is not normally distributed, and thus the second order polynomial trendline is not the best trendline to use for this data.

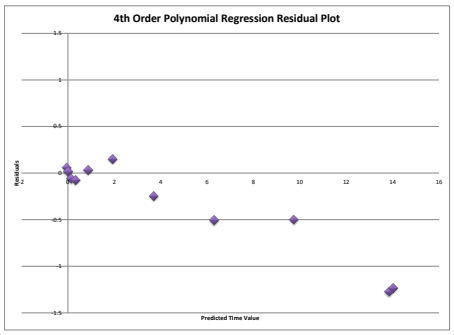
Likewise, the fourth order polynomial trendline does not fit the data well. Although there is more variance of positive and negative residuals, this is limited to time measurements less than 2 ms (Figure 4.2(c)on page 40). After that point, all residuals are negative for increasing values of elapsed time. Also, the distance of the



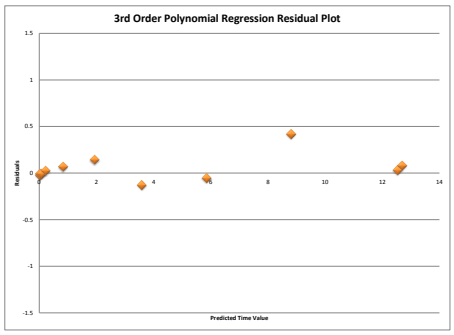
(a) A parabolic pattern in the residuals is easily observed.



(b) As the time value increases, the residual value (error) travels further from zero in the positive direction.



(c) As seen with the second order polynomial regression, the residual value (error) travels further from zero, though this time in the negative direction, as the time increases.



(d) The residual plot displays a mean closest to zero and a distribution that is most uniform out of the regression line types tested.

Fig. 4.2. Residual Plots of Linear and Polynomial Trendlines

residuals from zero only increases, further disqualifying the fourth order polynomial function from consideration.

The third order polynomial trendline exhibits residual behavior most indicative of a well-chosen line of best fit. Not only do the residuals have a mean closest to zero, they also do not form a distinct pattern (Figure 4.2(d) on page 40). Therefore, even

with the second lowest R-squared value, the third order polynomial trendline best exemplifies random error and thus fits the data better than the other trendlines. In conclusion, the empirical data confirms the calculated time complexity of $O(n^3)$ for the combination of matrix manipulations and execution of the Hungarian Algorithm.

Moving from the standalone environment, our research investigates on the performance of the combined algorithm with the PSO.

4.2 Code Profile

Performance measurement and analysis is reported in this section. The configuration of the computer used and test parameters will be introduced. Then, the data will be presented and analyzed. Finally, this research will recommend where the improvements can be made to the code.

4.2.1 Test Configuration

All testing was run on a Lenovo ThinkPad X220 laptop computer. The computational resources of this model include an Intel i5-2520M processor, 8GB of DDR3 memory at 1600MHz, a 250GB solid state drive (SSD), and integrated Intel HD Graphics 3000. The operating system on this machine is Windows 7 Professional 64-bit. All trials were conducted in release mode and were run on a single CPU thread without use of graphics processing resources.

4.2.2 Data Collection

The data collected in the code profile focuses on the time spent by particular sections of the code. Time trials were run on the functions that calculated the PSO, ran the Hungarian Algorithm, navigated the assets to new locations, and drew the GUI. Ten trials were run for 30 transmitters and 3, 5, 10, 15, 25, 50, and 75 assets. The trial times for each number of assets was averaged for each of the measured

categories; PSO, Hungarian Algorithm, Asset Navigation, and GUI. These averages are shown in Table 4.3 on page 42 and are graphed in Figure 4.3 on page 45.

Table 4.3.
Average Time (ms) in Various Functions for Different Asset Counts

Asset Number	**PSO**	**Hungarian Algorithm**	**Asset**	**GUI**
3	140.3113	0.0099	0.3793	6.64666
5	352.1408	0.00883	0.39975	6.32979
10	97.69465	0.02038	0.38156	7.75303
15	145.06875	0.03639	0.6925	8.70874
25	246.416	0.09249	0.56056	10.7546
50	509.891	0.3591	0.35342	14.4559
75	867.0009	0.89272	0.24767	21.33815

4.2.3 Results and Analysis

The first observation that should be made from this data is the efficiency of the Hungarian Algorithm. Usage of the algorithm alongside the PSO code results in time measurements that are very similar to those seen in the standalone environment. The time spent in the Hungarian Algorithm function does not even surpass 1 ms with the largest amount of assets, 75, tested. When compared to the sum of all the time measurements taken for each number of assets tested, the code spends an average of 0.037% of its time executing the Hungarian Algorithm. This portion of the time is so low in value that the visual comparison between it and the rest of the code can only be seen in Figure 4.4 on page 46

Like the Hungarian Algorithm, the asset navigation is not a significant contributor to the total time. On average, it consumes 0.213% of the total time to execute these functions. One aspect where the asset navigation differs from the Hungarian

Algorithm is how its execution time scales as the number of assets increases. The time in the Hungarian Algorithm steadily increases, whereas the time spent in asset navigation increases for asset counts up to 15, and then decreases. Our hypothesis for this behavior is that the PSO solution locations are spread out more for asset counts above 15. This decreases the distance, and thus the calculation and movement, needed for the assets to reach their assigned positions.

Of the three non-PSO related functions measured, the functions that draw the GUI contribute the most to the overall time. On average, the GUI will take 4.08% of the total time. Unlike the asset navigation, the time spent drawing the GUI continuously increases as the asset count increases. This is likely the result of the number of objects that need to be drawn steadily increases for increasing asset counts.

Lastly, the PSO calculation consumes the majority of the time measurements. Out of the total time of all functions, 95.671% on average will be dedicated to the PSO. Interestingly, the average PSO time for 10 assets is a third of the PSO time for 5 assets. A likely explanation for this behavior is that 10 assets are able to fulfill the majority of the constraints for most, if not all, of the transmitters. Therefore, the PSO does not have to spend as much time finding optimal solutions.

One can conclude that 10 assets is the most optimal number of assets to use in order to communicate with all 30 transmitters in the shortest amount of time. This behavior is least optimal with 50 and 75 assets, where average times to find a PSO solution are respectively 509.891 ms and 867.0009 ms. With this number of objects in the area, the frequency spectrum is very crowded. This negatively impacts the PSO as it has to spend more time minimizing interference between assets while finding solutions that maximize the fitness function.

4.2.4 Recommendations

As seen above, the PSO calculation time consumes the majority of the time in each interval. Therefore, optimization should be conducted in this area to reduce the

time expenditure and lower the global time spent by the application for every interval update. Likewise, methods to parallelize the PSO and harness a GPU for calculating solutions is another recommended approach to pursue in future work.

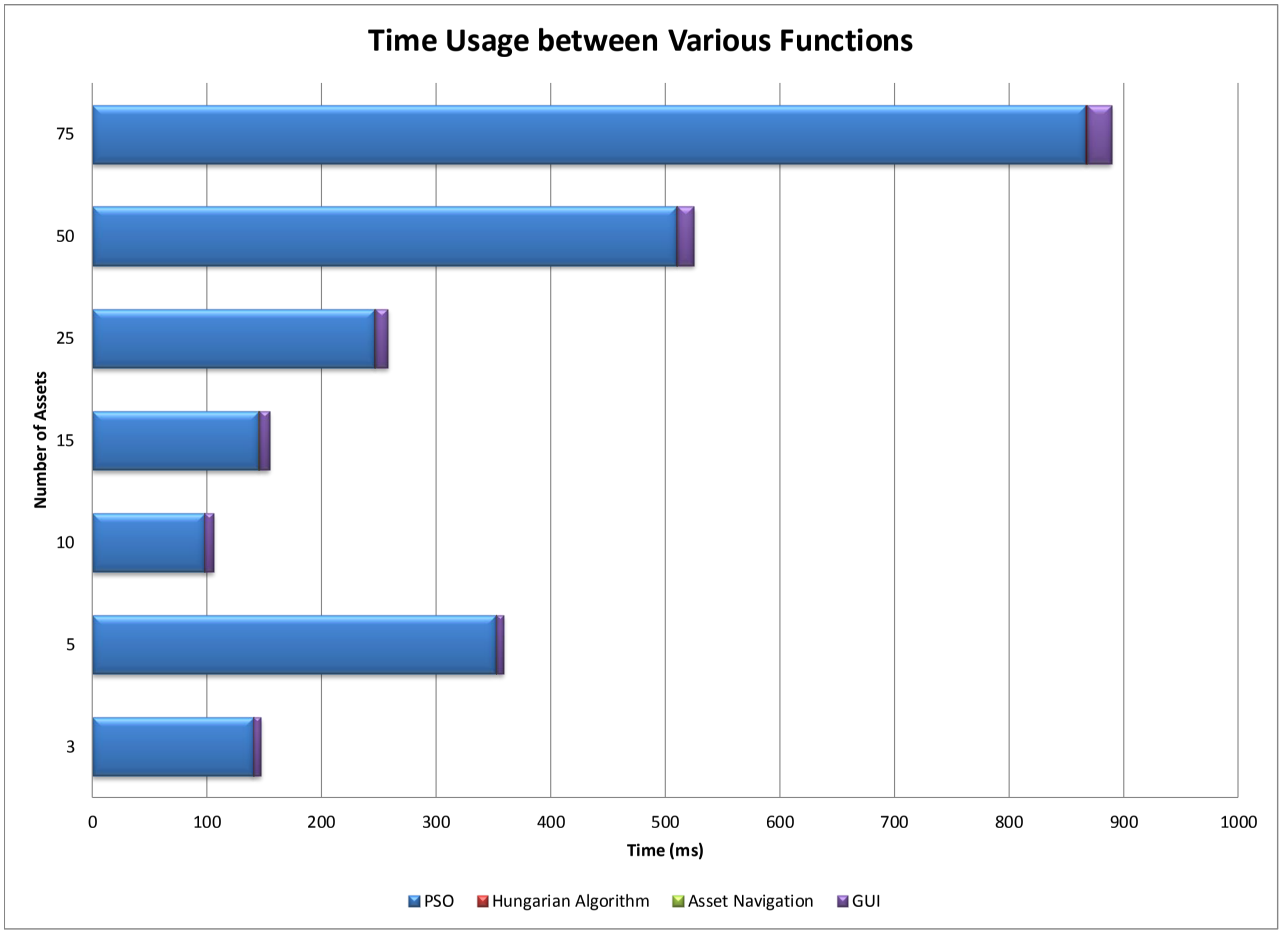


Fig. 4.3. A stacked bar graph visualizing the amount of time spent in code execution by four different areas.

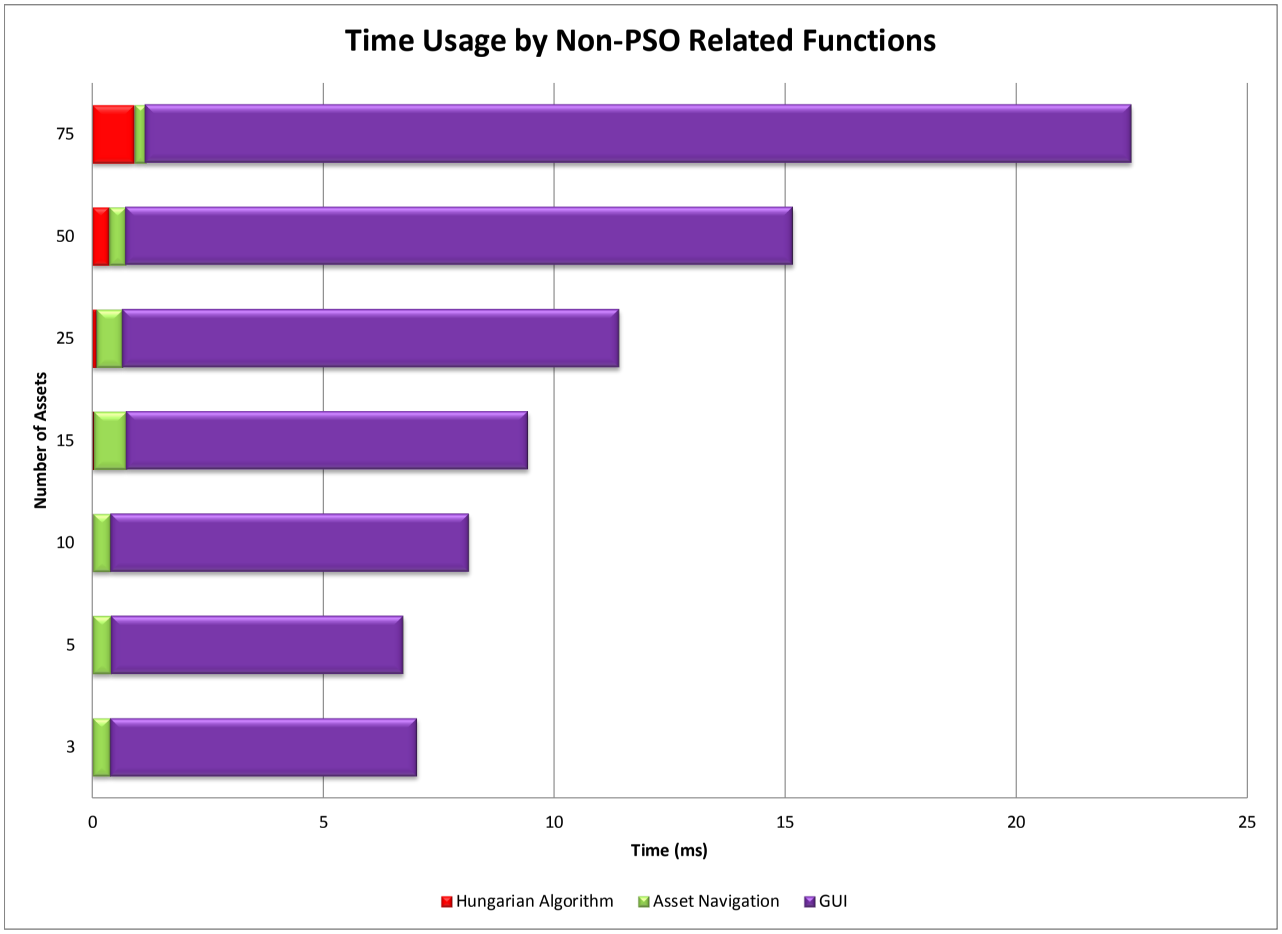


Fig. 4.4. A stacked bar graph focusing on the distribution of function not related to the execution of the PSO.

5. SUMMARY

5.1 Conclusions

As discussed in the abstract and introduction, this research makes improvements to the prior work in [2] and [3] to provide a dynamic asset allocation solution in less than one second. In doing so, the objectives introduced in the Introduction Chapter have been achieved. First, the behavior of the transmitters and assets was changed from static to dynamic. Also, the updates of the transmitters, assets, and PSO solution locations were automated. The user is able to start, pause, and reset the automatic updates that occur in the simulation. These updates occur in under one second, providing the warfighter with a real-time view of the EW operations in the battle space.

Next, this research accomplished the objective of how to assign assets to PSO solution locations. By modeling the relationship between assets and the PSO solution as a Bipartite Matching Assignment Problem, this mapping was achieved with a graph-based method. Known as the Hungarian Algorithm, the method runs in $O(n^3)$ time. The data provided in the Analysis chapter demonstrates the effectiveness of the combined algorithm implemented in this research, and its minimal impact on computation time. When running alongside the PSO and GUI functions, this implementation Hungarian Algorithm consumes no more than 0.1%, and on average 0.037%, of the total computation time.

The last objective, code profiling, was successfully completed in this research. The data collected from this profiling indicates that more than 95% of the time needed to update the information in the application is spent on the PSO. The profiling also demonstrated the drawing functions for the GUI consumed at most 7.325%, and 4.08% on average, of the total time. Even with an asset count of 75, the total computation

for the entire application remains under 1 second. However, the PSO's computational load can still be improved. This research provides recommendations on two different approaches to achieve this improvement.

5.2 Future Work

Further research can be conducted in three areas. The first area pertains to the transmitters, the second to the assets, and the third on the project as a whole.

5.2.1 Transmitters

The first way in which the transmitters can be improved is how they interact with the terrain and battle space as a whole. All transmitters are bound to the surface of the terrain for testing purposes in this research. However, in a real world scenario, some of these transmitters would have air capabilities, allowing them to depart from the terrain. Future work that addresses this possibility by unbinding the transmitters from the terrain surface would make the simulation of the battle space more realistic.

Second, this research focuses on dynamic position changes and velocities for the transmitters. In reality, the frequency behavior of each transmitter could change as well. Frequency hopping and other dynamic frequency behaviors should be included in future work. The changes in general frequency behavior caused by 3D movement of the transmitters should also be taken into consideration and pursued.

Lastly, an application programming interface (API) can be developed for this project. This research tests the performance of the PSO and assignment of assets to PSO solution locations. While the current implementation is satisfactory for analysis purposes, the addition of an API would make this application ready for real world use. The API would read in military information from different sources and use that information to update the positions and velocities of the transmitters. Further, the frequency characteristics for each transmitter can be modified with such an interface. These updates would be fed into the current application automatically, allowing the

program to run on actual data from the battle space without the user's manual input. Therefore, the dynamic 3D location and frequency behavior of the transmitters would provide effortless, real-time situational awareness of the battle space.

5.2.2 Assets

Unlike the transmitters, all of the assets possess a uniform speed. In the real world, different airborne assets may be used in conjunction with each other. For example, an F-16 fighter could be deployed alongside two Predator drones to support a squad of U.S. Marines. Both the fighter and the drones would have different speeds and thus would approach the locations of the PSO solution at different rates. Further, if there is a limitation on the altitude or distance a particular asset can travel, the assignment of assets to PSO solutions locations would have to adjust in order to maintain the EW operations being conducted.

In addition, assets used for EW may not always be airborne. EW operations can be launched from ground-based units. Also, EW could be expanded in the underwater battle space by using it with submarines or Unmanned Underwater Vehicles (UUV). Because the assets can be defined in terms of their role in air, land, or sea theaters of operation, their frequency characteristics would likely be different as well. Therefore, further work should investigate and provide a method that makes the frequency behavior of the assets dynamic and modifiable.

The API proposed in the above section detailing future work for the transmitters can also be utilized with the assets. The warfighter would experience many of the same benefits listed in the preceding section, allowing them to maximize their situation awareness with real-time data and updates.

5.2.3 Program Improvements

Several improvements can be made to the program as a whole. First, the variance in the PSO solution is high when the weights for the fitness function from the previous

research are used. This is likely a result of the recent changes that have been made to the code. Even with small changes in the positions of the transmitters, the PSO solution locations can jump as much as 100 kilometers from their previous location. This causes the assets to significantly change their current course, increasing the time and fuel spent to arrive at the final location. This delay and extra cost in fuel reduces the effectiveness and length of time of the EW operations that can be conducted. Because of this, the weights of the fitness function need to be adjusted in order to minimize the variance while still providing accurate PSO solutions.

Second, further analysis of the code profiling needs to be completed. Upon completion, the areas in which parallelization would benefit the operation and performance of the project would be identified. These areas would then be converted to run in parallel on a graphics processing unit (GPU) with the rest of the code running on the CPU.

Work on creating pheromones and implementing them into the rest of the program continues to be developed outside of the scope of this research. However, since the pheromones create areas of attraction and resistance for the assets, their interaction with the assets needs to be refined. One method of doing this will be to implement a path-finding algorithm that is able to avoid the pheromone areas of resistance completely while maintaining a course towards its assigned location. Consideration will also have to be made for whether the distance assets would have to travel to avoid these areas should be included in the distance calculated from the PSO solution locations.

REFERENCES

REFERENCES

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN 1995 - International Conference on Neural Networks*. Institute of Electrical and Electronics Engineers (IEEE).

[2] J. Reynolds, "Particle swarm optimization applied to real-time asset allocation (masters thesis)," Purdue University, Indianapolis. 2015.

[3] J. Crespo, "Asset allocation in frequency and in 3 spatial dimensions for electronic warfare application (masters thesis)," Purdue University, Indianapolis. 2016.

[4] L. Christopher, J. Reynolds, J. Crespo, R. Eberhart, and P. Shaffer, "Human fitness functions," in *Swarm/Human Blended Intelligence Workshop (SHBI), 2015*. IEEE, 2015, pp. 1–4.

[5] V. Mohan, J. G. Singh, and W. Ongsakul, "Sortino ratio based portfolio optimization considering evs and renewable energy in microgrid power market," *IEEE Transactions on Sustainable Energy*, vol. 8, no. 1, pp. 219–229, January 2017.

[6] A. Rhein, G. Balzer, R. Boya, and C. Eichler, "Multi-criteria optimization of maintenance and replacement strategies in transmission systems," in *2016 International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*, October 2016, pp. 1–6.

[7] R. Al-Hmouz, W. Pedrycz, A. Balamash, and A. Morfeq, "Granular representation schemes of time series: A study in an optimal allocation of information granularity," in *2013 IEEE Symposium on Foundations of Computational Intelligence (FOCI)*, April 2013, pp. 44–51.

[8] J. J. Liang and B. Y. Qu, "Large-scale portfolio optimization using multiobjective dynamic mutli-swarm particle swarm optimizer," in *2013 IEEE Symposium on Swarm Intelligence (SIS)*, April 2013, pp. 1–6.

[9] J. Dang, D. Edelman, R. Hochreiter, and A. Brabazon, "Swarm intelligence-based stochastic programming model for dynamic asset allocation," in *IEEE Congress on Evolutionary Computation*, July 2010, pp. 1–8.

[10] X. Zhang and K. Zhang, "Application of adaptive particle swarm optimization in portfolio selection," in *2009 First International Conference on Information Science and Engineering*, December 2009, pp. 3977–3980.

[11] V. Parque, S. Mabu, and K. Hirasawa, "Guided genetic relation algorithm on the adaptive asset allocation," in *SICE Annual Conference 2011*, September 2011, pp. 173–178.

[12] D. P. Bertsekas, M. L. Homer, D. A. Logan, S. D. Patek, and N. R. Sandell, "Missile defense and interceptor allocation by neuro-dynamic programming," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 30, no. 1, pp. 42–51, January 2000.

[13] G. Arslan, J. D. Wolfe, J. Shamma, and J. L. Speyer, "Optimal planning for autonomous air vehicle battle management," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 4, December 2002, pp. 3782–3787 vol.4.

[14] J. McDonnell, A. Rice, A. Spydell, and S. Stremler, "Dynamic tactical air strike asset allocation using evolutionary computation," in *2005 IEEE Congress on Evolutionary Computation*, vol. 1, September 2005, pp. 810–815 Vol.1.

[15] G. V. Avvari, D. Sidoti, M. Mishra, L. Zhang, B. K. Nadella, K. R. Pattipati, and J. A. Hansen, "Dynamic asset allocation for counter-smuggling operations under disconnected, intermittent and low-bandwidth environment," in *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, May 2015, pp. 1–6.

[16] W. An, D. F. M. Ayala, D. Sidoti, M. Mishra, X. Han, K. R. Pattipati, E. D. Regnier, D. L. Kleinman, and J. A. Hansen, "Dynamic asset allocation approaches for counter-piracy operations," in *2012 15th International Conference on Information Fusion*, July 2012, pp. 1284–1291.

[17] E. Raboin, P. vec, D. Nau, and S. K. Gupta, "Model-predictive target defense by team of unmanned surface vehicles operating in uncertain environments," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 3517–3522.

[18] P. R. E. Cutler and X. T. Nguyen, "Description of a rule-based model for the automatic allocation of airborne assets," in *Sixth International Conference of Information Fusion, 2003. Proceedings of the*, vol. 2, July 2003, pp. 979–985.

[19] M. G. Oxenham and P. Cutler, "Accommodating obstacle avoidance in the weapons allocation problem for tactical air defence," in *2006 9th International Conference on Information Fusion*, July 2006, pp. 1–8.

[20] A. Preece, D. Pizzocaro, K. Borowiecki, G. de Mel, M. Gomez, W. Vasconcelos, A. Bar-Noy, M. P. Johnson, T. L. Porta, H. Rowaihy, G. Pearson, and T. Pham, "Reasoning and resource allocation for sensor-mission assignment in a coalition context," in *MILCOM 2008 - 2008 IEEE Military Communications Conference*, November 2008, pp. 1–7.

[21] G. A. Mills-Tettey, A. Stentz, and M. B. Dias, *The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs*, 2007.

[22] C. J. Huang, "Integrate the Hungarian method and genetic algorithm to solve the shortest distance problem," in *2012 Third International Conference on Digital Manufacturing Automation*, July 2012, pp. 496–499.

[23] Y. Zhang, G. Song, G. Qiao, Z. Li, Y. Li, and A. Song, "Strategy research of role assignment and formation control for multi-robot systems," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, December 2013, pp. 958–963.

[24] M. Zhao, Y. Zuo, Y. Y. Fang, and M. D. Li, "Sensor assignment method based on time-varying measurement variance for tracking multi-targets," in *2016 Chinese Control and Decision Conference (CCDC)*, May 2016, pp. 3368–3372.

[25] W. Liu, C. Wen, and G. Luo, "Multi-sensor joint configuration and tracking algorithm - fixed sensor systems," in *Proceedings of the 32nd Chinese Control Conference*, July 2013, pp. 4576–4581.

[26] Z. Liao, J. Wang, S. Zhang, J. Cao, and G. Min, "Minimizing movement for target coverage and network connectivity in mobile sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 7, pp. 1971–1983, July 2015.

[27] D. Zhang and L. Wang, "Target topology based task assignment for multiple mobile robots in adversarial environments," in *2007 46th IEEE Conference on Decision and Control*, December 2007, pp. 5323–5328.

[28] D. Turra, L. Pollini, and M. Innocenti, "Fast unmanned vehicles task allocation with moving targets," in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, vol. 4, December 2004, pp. 4280–4285 Vol.4.

[29] M. Alighanbari, Y. Kuwata, and J. P. How, "Coordination and control of multiple UAVs with timing constraints and loitering," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 6, June 2003, pp. 5311–5316 vol.6.

[30] X. Han, M. Mishra, S. Mandal, H. Bui, D. F. M. Ayala, D. Sidoti, K. R. Pattipati, and D. L. Kleinman, "Optimization-based decision support software for a team-in-the-loop experiment: Multilevel asset allocation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 8, pp. 1098–1112, August 2014.

[31] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, March 1955.

[32] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, March 1957.

[33] M. Golin, "Bipartite matching and the Hungarian method," August 30, 2006 (accessed December 21, 2016). [Online]. Available: http://www.cse.ust.hk/~golin/COMP572/Notes/Matching.pdf

[34] D. Bruff, "The assignment problem and the Hungarian method," 2005 (accessed December 21, 2016). [Online]. Available: http://math.harvard.edu/archive/20_spring_05/handouts/assignment_overheads.pdf

[35] T. H. Cormen and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001.

[36] M. Golin, "Maximum flow," September 9, 2004 (accessed February 25, 2016). [Online]. Available: http://home.cse.ust.hk/~golin/COMP572/Notes/max_flow.pdf

[37] "Assignment problem and Hungarian algorithm," https://www.topcoder.com/community/data-science/data-science-tutorials/assignment-problem-and-hungarian-algorithm/#!, accessed: December 21, 2016.

[38] G. Dass, "Regression analysis: How do I interpret r-squared and assess the goodness-of-fit?" https://www.linkedin.com/pulse/regression-analysis-how-do-i-interpret-r-squared-assess-gaurhari-dass, accessed: March 20, 2016.

[39] "To err is human, to err randomly is statistically divine," https://www.qualitydigest.com/inside/quality-insider-article/why-you-need-check-your-residual-plots-regression-analysis.html, accessed: March 20, 2016.

[40] "Interpreting residual plots to improve your regression," http://docs.statwing.com/interpreting-residual-plots-to-improve-your-regression/#hetero-header, accessed: March 20, 2016.

[41] G. R. Waissi, *Applied Statistical Modeling*, 2nd ed. Arizona State University, 2015.